Project 2

A* Implementation

Gabriel Sturtevant

Comp 282

Dr. Barnes

Monday- Wednesday 9:30 - 10:45

The goal of the second project was to implement a priority queue, and a graph in the utilization of the A* algorithm. We needed to be able to import a text file that defined a series of 'waypoints' and their attributes, including which other 'waypoints' are linked to it, and place those waypoints onto the screen through the SimulationFramework with a marker of the correct size and color . We are also expected to capture mouse clicks corresponding to the start waypoint and another corresponding to the ending waypoint. Our solution then needs to utilize a version of the A* algorithm in order to find a path to the end waypoint. Another caveat of our design is that, once the we have found the path, our bot is supposed to then walk the given path. If while traversing the path our bot finds a map and isn't already following a map to it's destination, the bot is then supposed to find a path to the treasure, traverse to the treasure and then continue to the original destination in the same manner.

The Map class was originally chosen to be the central location to hold any collections of WayPoints, as such this was the logical location to have the A* algorithm, since it uses a separate HashMap and PriorityQueue. Since the A* algorithm was going into the Map class, it seemed appropriate to also place the methods associated with movement and returning the path in this class as well. In addition to the the rest of the methods in the Map.java file, there is an additional class compareTo which implements comparator, and overrides the the compare() method used by the PriorityQueue.

The A* algorithm and the goTo method were designed and implemented in such a way that if they need to run, they will only run once per itteration of the SimulateAlgorithm method in my Sturtevant282P2 class. The fact that they need to be run multiple times in row, necessitated them to be designed so that they are controlled by boolean flags. If while looking for the destination, the A* algorithm opens a WayPoint that is the same as the destination WayPoint, the algorithm will set a flag stating that the path has been found, then call a method that will push the current waypoint into a FILO stack before assigning the value of current WayPoint to parent Waypoint of the top WayPoint in the stack. It continues this way until the top WayPoint on the stack is the same as the destination Waypoint.

Once this occurs, another flag is set letting the program know that the path has been returned and that it can begin traversing the path. The bot traverses the path, and only stops if it has reached it's destination or a map. If it finds a map, it stops and recalculates a path leading to the map's treasure, unless the bot has already visited the WayPoint that that map leads to, in which case it ignores the map and continues on. The other halting condition is if the bot has reached it's destination. If the bot is at it's destination and that destination was the original destination, not just the location of gold that it got from a map, the simulation will end. If the destination is just a the location of gold from a map, the program resets the different data structures it uses for path finding and traversing a path and recalculates it's ending path based on the original end WayPoint and also recalculates the heuristic values of each WayPoint based on the new destination.

The Waypoint class contains data about each location on the map, and is also does calculations such as distances, and it also keeps track of traversal costs for each WayPoint from the beginning WayPoint.

The Player class is just used to keep track track of and modify information about the player such as the player's wealth and strength.